# LANGUAGE GUIDE FOR COMPLEX EXPRESSIONS

Version 0.3

January 16, 2010

## Introduction

This language is a subset of the C programming language. There are some language differences as noted in the detail section. Any operation that accepts a complex number can also accept a complex expression.

## Predefined Variables

Some variables are pre-defined and are used as an interface between the language and the application. The following variables are pre-defined: z, p1, cp, lastsqr, maxit, n, test, whitesq.

z:

z is the main complex variable and is usually iterated. For a Julia type expression, z is initialized to the complex value associated with the current pixel on the screen. For a mandelbrot type expression, z is initialized to either zero or a complex constant. The magnitude of z is tested to see if it is larger than bailout. If so, the iteration is stopped and z is declared to have escaped. The number of iterations required before z escapes is used as an index into a color table to determine the color the pixel is painted.

Summary:

Julia: z0 = cp; z = z * z + c; // Common Julia Fractal of z squared

Mandelbrot: z0 = c; z = z * z + cp; // Common Mandelbrot Fractal of z squared

Stop iterating z if mag(z) >= bailout

p1:

p1 is a complex constant whose value can be changed on the user interface.

cp:

cp is a complex number that represents the complex value associated with a pixel on the display.

lastsqr:

lastsqr is the magsqr of the arguments of the last sqr. It is needed to be compatible with some imported parameter files.

maxit:

maxit is the maximum number of iterations. It is needed to be compatible with some imported parameter files.

n:

n is the iteration number of the loop. n is zero based so the first time through the loop n = 0. The value of n can be tested and used with a conditional expression to initialize variables. The expression z = n ? z : 0; will set z = 0 on the first time through the loop and leave z unaffected on subsequent iterations.

test:

test is the escape test result for the fractal. If test is false, the fractal is declared escaped and the loop stops. If test is true the loop continues as long as the maximum number of iterations has not been exceeded. Use the magsqr function (magnitude squared function) for faster execution than the mag function.

Example: test = magsqr(z) < 4;

whitesq:

whitesq is (row + column) modulo 2. It is usually used to blend two images together and is included for compatibility with some imported parameter files.

## User Defined Variables

To add a user defined variable, simply use it in an equation. You should set its value before you read it's value.

## Initialization of Variables

Statements before the @beginloop; compiler directive are only executed on the first iteration of the loop.

Example:

z =0; // Initializes z to 0

@beginloop; // Start of the loop

z = z * z + cp; // Execute until test is false or maxIter is reached

test = magsqr( z ) <= 4; // Stop if |z| squared greater than 4

## Constants

Constants are complex double precision. Exponents are IEEE compliant. To enter a complex constant use the complex constant function or its shortcut (a,b) as shown in the example below.

Examples:

a = -3.14e-10; b = 4E10; c = 3;

b = ( 2 , 4 ); // b is 2 + i4

c =complex( 2 , 4 );  // c is 2 + i4

## Command Summary

1. Arithmetic: + - * / ^ %
2. Assignment: =
3. Comment: // /* */
4. Compare: < > <= >= == != && ||
5. Conditional Expression: ?:
6. Functions: abs acos acosh asin asinh atan atanh ceil conj cos cosh exp flip floor imag log log10 mag magsqr pow real sin sinh sqrt tan tanh

## Command Detail

In all of the descriptions below a and b are complex numbers. Each has a real part and an imaginary part. So, the variable a could be represented as (real(a) , imag(a)) or (a.r , a.i).

## Arithmetic

- +

Adds two complex numbers a and b and returns a complex number.

```
a + b
real = a.r + b.r
```

imag = a.i + b.i

Example: c = a + b;

- -

Subtracts two complex numbers a and b and returns a complex number.

a - b
real = a.r - b.r
imag = a.i - b.i

Example: c = a - b;

Unary minus negates a variable or parenthesized expression.

Example: b = -a; b = -(a * 2);

- *

Multiplies two complex numbers a and b and returns a complex number.

a * b
real = a.r * b.r - a.i * b.i
imag = a.r * b.i + a.i * b.r

Example: c = a * b;

- /

Divides two complex numbers a and b and returns a complex number.

a / b
real = (a.r * b.r + a.i * b.i) / (b.r * b.r + b.i * b.i)
imag = (a.r * b.i - a.i * b.r) / (b.r * b.r + b.i * b.i)

Example: c = a / b;

- ^

Raises the complex number a to the power of the complex number b and returns a complex number. The operator ^ is right associative so a ^ b ^ c = a ^ (b ^ c). The c language uses pow(a , b) instead of the ^ operator. Fractint does not appear to be right associative.

```
a ^ b
temp1 = log(a.r * a.r + a.i * a.i) / 2
temp2 = atan2(a.i , a.r)
temp3 = temp1 * b.r - temp2 * b.i
temp4 = temp2 * b.r + temp1 * b.i
temp5 = exp( temp3 )
real = temp5 * cos( temp4 )
imag = temp5 * sin( temp4 )
```

Example: c = a ^ b;

- *%*

Performs the scalar modulus operation on the real part of a complex number. For the complex modulus see the mag function. Returns 0 for b <= 0.

```
a % b
if (b > 0) real = a.r % b.r
else real = 0
imag = 0
```

Example c = a % b;

## Assignment:

- =

The equal copies the value of  a complex expression to a complex variable. Every expression must start with a variable followed by  = .

a = b

Example: a = b;

## Comment:

- //

Any time // is encountered the rest of the line is ignored by the compiler.

Examples:

```
// This line is a comment
a = b + c; // Everything to the right of // is a comment
```

- /*

The start of a block comment.

- */

The end of a block comment.

Example:

```
/* All of these three lines
a = b * c;
are comments */
```

## Compare:

- <

a < b returns 1.0 + 0.0i if a.r is less than b.r. Otherwise, it returns a 0.0 + 0.0i.

Example c = a < b;

- >

a > b returns 1.0 + 0.0i if a.r is greater than than b.r. Otherwise, it returns a 0.0 + 0.0i.

Example c = a > b;

- >

a > b returns 1.0 + 0.0i if a.r is greater than than b.r. Otherwise, it returns a 0.0 + 0.0i.

Example c = a > b;

- <=

a <= b returns 1.0 + 0.0i if a.r is less than or equal to b.r. Otherwise, it returns a 0.0 + 0.0i.

Example c = a <= b;

- >=

a >= b returns 1.0 + 0.0i if a.r is greater than or equal to b.r. Otherwise, it returns a 0.0 + 0.0i.

Example c = a >= b;

- ==

a == b returns 1.0 + 0.0i if a.r is equal to b.r. Otherwise, it returns a 0.0 + 0.0i. Numbers that differ by less than 1E-6 are considered equal.

Example c = a == b;

- !=

a != b returns 1.0 + 0.0i if a.r is not equal to b.r. Otherwise, it returns a 0.0 + 0.0i. Numbers that differ by less than 1E-6 are considered equal.

Example c = a != b;

- &&

a && b returns 1.0 + 0.0i if both a.r and b.r are greater than 0. Otherwise, it returns a 0.0 + 0.0i.

Example c = real(a) > 1 && imag(a) < 0;

- ||

a || b returns 1.0 + 0.0i if either a.r or b.r are greater than 0. Otherwise, it returns a 0.0 + 0.0i.

Example c = real(a) > 1 || imag(a) < 0;

## Conditional Expression:

- ?:

The conditional expression  a ? b : c returns b if a.r > 0 otherwise it returns c. This differs from the C language in that all of the expressions are evaluated whether they are used or not.

Example: z = real(z) >= 0 ? (z - 1) * c : (z + 1) * c;  // Barnsleyj1

## Functions:

- abs

Returns the absolute value of the individual components of a complex number. See mag() for the complex modulus, complex absolute value, or complex magnitude. For the modulus of the real part only see % in the arithmetic section.

abs( a )
real = abs( a.r )
imag = abs( a.i )

Example: b = abs( a );

- acos

Returns the arccosine of a complex number.

```
acos(a)
temp1 = sqrt((a.r + 1) * (a.r + 1) + a.i * a.i) / 2
temp2 = sqrt((a.r - 1) * (a.r - 1) + a.i * a.i) / 2
temp3 = temp1 + temp2
temp4 = temp1 - temp2
real = acos( temp4 )
imag = -log(temp3 + sqrt(temp3 * temp3 -1)) for a.i >= 0
imag = log(temp3 + sqrt(temp3 * temp3 -1)) for a.i < 0
```

Example: b = acos(a);

acosh

Returns the hyperbolic arccosine of a complex number.

- acosh(a)

```
temp3 = a.r * a.r - a.i * a.i - 1;
temp4 = a.i * a.r + a.r * a.i;
temp5 = atan2(temp4 , temp3) / 2.0;
temp2 = sqrt(sqrt(temp3 * temp3 + temp4 * temp4));
temp1 = temp2 * cos( temp5 ) + a.r;
temp2 = temp2 * sin( temp5 ) + a.i;
a.r = log(temp1 * temp1 + temp2 * temp2) / 2.0;
a.i = atan2(temp2 , temp1);
```

Example: b = acosh(a);

- asin

Returns the arcsine of a complex number.

```
asin(a)
temp1 = sqrt((a.r + 1) * (a.r + 1) + a.i * a.i) / 2
temp2 = sqrt((a.r - 1) * (a.r - 1) + a.i * a.i) / 2
temp3 = temp1 + temp2
temp4 = temp1 - temp2
real = asin( temp4 )
imag = log(temp3 + sqrt(temp3 * temp3 -1)) for a.i >= 0
imag = - log(temp3 + sqrt(temp3 * temp3 -1)) for a.i < 0
```

Example: b = asin(a);

• asinh

Returns the hyperbolic arcsine of a complex number.

```
temp3 = a.r * a.r - a.i * a.i + 1;
temp4 = a.i * a.r + a.r * a.i;
temp5 = atan2(temp4 , temp3) / 2.0;
temp2 = sqrt(sqrt(temp3 * temp3 + temp4 * temp4));
temp1 = temp2 * cos( temp5 ) + a.r;
temp2 = temp2 * sin( temp5 ) + a.i;
a.r = log(temp1 * temp1 + temp2 * temp2) / 2.0;
a.i = atan2(temp2 , temp1);
```

Example: b = asinh(a);

• atan

Returns the arctangent of a complex number.

```
atan(a)
real = atan2(2 * a.r , 1 - a.r * a.r - a.i * a.i) / 2
imag = log((a.r * a.r + (a.i + 1) * (a.i + 1)) / (a.r * a.r + (a.i - 1) * (a.i - 1))) / 4
```

Example: b = atan(a);

• atanh

```
atanh(a)
temp4 =  1 + a.r
temp5 = 1 - a.r
temp1 = temp4 * temp5 - a.i * a.i;
temp2 = a.i * temp5 + temp4 * a.i;
temp3 = temp5 * temp5 + a.i * a.i;
temp1 = temp1 / temp3;
temp2 = temp2 / temp3;
real = log(temp1 * temp1 + temp2 * temp2) / 4.0;
imag = atan2(temp2 , temp1) / 2.0;
```

Example: b = atanh(a);

To do a two argument complex arctangent, use the following equation:
 - i * log((real(a) + i * imag(b)) / sqrt(real(a) * real(a) + imag(a) * imag(a)))
where a and b are the two complex arguments.

- ceil

Returns the ceiling of the individual components of a complex number.

ceil(a)
real = ceil( a.r )
imag = ceil( a.i )

Example: b = ceil( a );

- conj

Returns the complex conjugate of a complex number. The complex conjugate flips the sign of the complex part.

conj(a)
real = a.r
imag = -a.i

Example: b = conj( a );

- cos

Returns the complex cosine of a complex number.

cos( a )
real = cos( a.r ) * cosh( a.i )
imag = -sin( a.r ) * sinh( a.i )

Example: a = cos( b / 2 );

- cosh

Returns the complex hyperbolic cosine of a complex number.

cosh( a )
real = cosh( a.r ) * cos( a.i )
imag = sinh( a.r ) * sin( a.i )

Example: a = cos( b / 2 );

- exp

Raises e to the power of a complex number.

exp( a )
temp1 = exp( a.r )
real = temp1 * cos( a.i )
imag = temp1 * sin( a.i )

Example: b = exp( a );

- flip

Returns the complex number with the real and imaginary parts flipped.

flip(a)
real = a.i
imag = a.r

Example: b = flip( a );

- floor

Returns the floor of the individual components of a complex number.

floor(a)
real = floor( a.r )
imag = floor( a.i )

- imag

Returns the imaginary portion of a complex number as a real complex number.

imag( a )

real = a.i
imag = 0

Example b = imag( a );

- log

Returns the natural logarithm of a complex number.

log( a )
real = log(a.r * a.r + a.i * a.i) / 2
imag = atan2(a.i , a.r)

Example: b = log( a );

- log10

Returns the natural logarithm of a complex number.

log10( a )
temp1 = log(10)
real = log(a.r * a.r + a.i * a.i) / (2 * temp1)
imag = atan2(a.i , a.r) / temp1

Example: b = log10( a );

- mag

Returns the real magnitude of a complex number. This operation is sometimes referred to as the complex modulus or complex absolute value. In math formulas it is often written as |a|.

mag(a)
real = sqrt(a.r * a.r + a.i * a.i)
imag = 0

- magsqr

Returns the real magnitude of a complex number. This operation is sometimes referred to as the complex modulus squared or complex absolute value squared. In math formu-

|as it is often written as |a|^2. This function executes faster than the mag function since it has no square root.

```
magsqr(a)
real = a.r * a.r + a.i * a.i
imag = 0
```

• pow

Raises the complex number a to the power of the complex number b and returns a complex number.

```
pow(a , b)
temp1 = log(a.r * a.r + a.i * a.i) / 2
temp2 = atan2(a.i , a.r)
temp3 = temp1 * b.r - temp2 * b.i
temp4 = temp2 * b.r + temp1 * b.i
temp5 = exp( temp3 )
real = temp5 * cos( temp4 )
imag = temp5 * sin( temp4 )
```

Example: c = pow(a , b);

• real

Returns the real portion of a complex number.

```
real( a )
real = a.r
imag = 0
```

Example b = real( a );

• sin

Returns the complex sine of a complex number.

```
sin( a )
real = sin( a.r ) * cosh( a.i )
imag = cos( a.r ) * sinh( a.i )
```

Example: a = sin( b );

- sinh

Returns the complex hyperbolic sine of a complex number.

```
sinh( a )
real = sinh( a.r ) * cos( a.i )
imag = cosh( a.r ) * sin( a.i )
```

Example: a = sinh( b );

- sqrt

Returns the square root of a complex number.

```
sqrt(a)
temp1 = sqrt(a.r * a.r + a.i * a.i)
real = sqrt(fabs((a.r + temp1) / 2))
if(a.i >= 0) imag = sqrt(fabs((temp1 - a.r) / 2))
else imag = -sqrt(fabs((temp1 - a.r) / 2))
```

- tan

Returns the complex tangent of a complex number.

```
tan( a )
temp1 = 2 * a.r
temp2 = 2 * a.i
temp3 = cos( temp1 ) + cosh( temp2)
real = sin( temp1 ) / temp3
imag = sinh( temp2 ) / temp3
```

Example: a = tan( b );

- tanh

Returns the complex hyperbolic tangent of a complex number.

```
tanh( a )
temp1 = 2 * a.r
temp2 = 2 * a.i
```

temp3 = cosh( temp1 ) + cos( temp2)
real = sinh( temp1 ) / temp3
imag = sin( temp2 ) / temp3

Example: a = tanh( b );

## Precedence:

- ( )

Changes the precedence of an expresssion.

Example: a = 2 * (1 + 3);  // a is 8